ELEC 204 Digital Design Project Report

Project Partners & ID Numbers: İsmail Ozan Kayacan (69103) - Barış Kaplan (69054)

Date: 06/09/2021

1. Introduction and objectives

In our project, we have implemented the VHDL code of a statistical calculator which calculates the mean, mode, median, and range of 4 unsigned 4-bit binary numbers depending on the value of the input called modeSelect.

MAIN CONCEPTS & THEIR DEFINITIONS

The definition of mean: The mean is the value which is obtained by dividing the summation of a set of numbers by the amount of the numbers in this set.

Example set: 0000, 0001, 0010 (0000 is equal to 0 in decimal. 0001 is equal to 1 in decimal. 0010 is equal to 2 in decimal. The summation of 0,1, and 2 is equal to 3. There are 3 numbers in the given set here. So, the mean of the numbers in this set is 3/3, which is 1 in decimal and which is 0001 in binary)

The definition of mode*:* The mode is the number which appears at most in a number set.

Example set: 0001,1111,0001,0101 (Since 0001 is the binary number which repeats itself at most, 0001 is the mode of this set.)

The definition of median: If the total number of elements in a set which is sorted in an increasing order is odd, the median is equal to the element in the middle of the set. If the total number of elements in a set is even, the median is equal to the average of the summation of the number right after the middle of the set and the number right before the middle of the set.

Example set-1: 0000,0001,0011,0100 (The total number of elements in this set is even. Hence, the median is equal to the average of the summation of 0001 and 0011. 0001 is 1 in decimal. 0011 is 3 in decimal. So, for the given example set here, the median is 4/2, which is 2.)

Example set-2: 0000, 0001, 0010 (The total number of elements in this set is odd. So, the median is equal to the middle element, which is 0001, in the set. For this example set, the median is equal to 1 in decimal.)

The definition of range: The range is the difference between the maximum number and the minimum number in the set.

Example set: 0000, 0001, 0010 (In this set, the maximum number is equal to 0010. Moreover, the minimum number is equal to 0000. So, the range of this set is equal to 0010-0000, which is equal to 0010. For this example set, the range is equal to 2 in decimal and 0010 in binary)

The objectives of this project

Getting better, refreshing, and improving our knowledge and abilities that we gained in labs and lecture during the semester such as:

1-) Implementing the VHDL code of full adder.
2-) Using ‘port map’ in the VHDL code.

3-) Implementing the VHDL code of the unsigned comparator.
4-) Using the clock synchronized processes in VHDL.
5-) Using the unsigned data types. (incrementing or decrementing unsigned data types in the VHDL code.)
6-) Defining a process in the VHDL code.
7-) Defining and using signals which are INTEGER type.
8-) Using the component of a VHDL code in another VHDL code.
9-) Using if and elsif statements in a processs.
10-) Using when else statement in the VHDL code.

What our code has to do

Our program should take 4 4-bit binary numbers called num1, num2, num3, num4 and should calculate the mean, mode, median, and range of these numbers, and should display one of them in the leds depending on the value of the modeSelect.

We used 10 different VHD files for our program. Their names, how they work and their input-outputs are as following:

MAIN.VHD

This is the main source code of the program, it's working principle will be explained in detail in the next pages.

The inputs of Main.vhd

There are 10 inputs in Main.vhd. The names of the inputs are clk, switch0, switch1, switch2, switch3, switch8, switch9, num1EnteredSwitch, num2EnteredSwitch, num3EnteredSwitch, and num4EnteredSwitch all of which are 1 bit STD_LOGIC signals. Switch0, switch1, switch2 and switch3 decides the value of each numbers bits from LSB to MSB. Switch8 and switch9 decides the value of modeSelect from LSB to MSB. num1EnteredSwitch, num2EnteredSwitch, num3EnteredSwitch, and num4EnteredSwitch save the inputted binary numbers.

The outputs of Main.vhd

There is 1 output in the Main.vhd. The name of the output is result which is 4-bit STD_LOGIC_VECTOR. The output called result is used to show possible outputs which are mean, mode, median, and range.

COMPARATOR.VHD

Comparator, starting from most significant bits, compares two numbers, and gives output GAB=1 if A is greater than B, 0 otherwise.

The inputs of Comparator.vhd

There are 2 inputs in the Comparator.vhd. The names of the inputs are A and B. which are 4-bit STD_LOGIC_VECTOR signals.

The outputs of Comparator.vhd

There is 1 output in the Comparator.vhd. The output is called GAB. GAB is a 1-bit output. If A is greater than B, GAB will be equal to 1. The type of the output called GAB is STD_LOGIC.

ONEBITADDER.VHD

Onebitadder, takes two input numbers and carry in, and calculates their sum and carry out with logic gates by full adder implementation.

The inputs of oneBitAdder.vhd

There are 3 inputs in oneBitAdder.vhd. The names of the inputs are oneBitCin, oneBitNum1, and oneBitNum2 which are STD_LOGIC. oneBitCin is the the 1-bit carry in which comes to the addition of the oneBitNum1 and oneBitNum2. The inputs called oneBitNum1 and oneBitNum2 are the numbers which are added in oneBitAdder.vhd.

The outputs of oneBitAdder.vhd

There are 2 outputs in oneBitAdder.vhd. The names of the outputs are oneBitResult and oneBitCout which are STD_LOGIC. The output called oneBitResult is the result of the addition of oneBitNum1 and oneBitNum2. The output called oneBitCout is the carry out at the end of the addition.

ADDER4BIT4BIT.VHD

Adder4bit4bit using oneBitAdder as a component, calculates the sum of two 4-bit binary numbers. It applies oneBitAdder to each bit one by one.

The inputs of adder4bit4bit.vhd

There are 3 inputs in adder4bit4bit.vhd. The inputs are called cin, num1, and num2. The input called cin is 1-bit. The input called num1 is 4-bit. Moreover, the input called num2 is 4-bit. The input called cin is the carry in which will be added to the summation of the zeroth bit of num1 and the zeroth bit of num2. The inputs called num1 and num2 are the numbers which will be added. The type of the cin is STD_LOGIC. The type of the input called num1 is STD_LOGIC_VECTOR, and the type of the input called num2 is STD_LOGIC_VECTOR.

The outputs of adder4bit4bit.vhd

There are 2 outputs in adder4bit4bit.vhd. The outputs are called Cout and result. The output called Cout is 1-bit. The type of the output called Cout is STD_LOGIC. The output called result is 7-bit. The type of the output called result is STD_LOGIC_VECTOR. The output called Cout is the carry which goes out after each of the addition of the bits of 4-bit binary numbers. The output called result is the addition of 4-bit unsigned binary numbers.

ADDER7BIT4BIT.VHD

Adder7bit4bit using oneBitAdder as a component, calculates the sum of 7-bit and 4-bit binary numbers. It applies oneBitAdder to each bit one by one.

The inputs of adder7bit4bit.vhd

There are 3 inputs in adder7bit4bit.vhd. The names of these inputs are cin, num1, and num2. The input called cin is a 1-bit input. The input called num1 is a 7-bit input. Furthermore, the input called num2 is a 4-bit input. The type of the input called num1 is STD_LOGIC_VECTOR, and the type of the input called num2 is STD_LOGIC_VECTOR. The input called cin is the carry in which will be added to the summation of the zeroth bit of num1 and the zeroth bit of num2. The inputs called num1 and num2 are the numbers which will be added.

The outputs of adder7bit4bit.vhd

There are 2 outputs in adder7bit4bit.vhd. The outputs are called Cout and result. The output called Cout is 1-bit. The type of the output called Cout is STD_LOGIC. The output called result is 7-bit. The type of the output called result is STD_LOGIC_VECTOR. The output called Cout is the carry which goes out after each addition of the bits of 7-bit unsigned binary number and 4-bit unsigned binary number. The output called result is the addition of this 7-bit unsigned binary number and this 4-bit unsigned binary number.

DIVIDERBY2.VHD

DividerBy2 divides the binary number by 2 by shifting the digits to the right by 1.

The inputs of dividerBy2.vhd

There is 1 input in dividerBy2.vhd. This input is called num. The type of this input is STD_LOGIC_VECTOR. Moreover, this input is 5-bit. The input called num is the binary number which is divided by 2 in the dividerBy2.vhd.

The outputs of dividerBy2.vhd

There is 1 output in dividerBy2.vhd. This output is called result which is 4-bit STD_LOGIC_VECTOR. The output called result is the resulting binary number after dividing the input called num by 2.

DIVIDERBY4.VHD

DividerBy4 divides the binary number by 4 by shifting the digits to the right by 2.

The inputs of dividerBy4.vhd
There is 1 input in dividerBy4.vhd. This input is called num. The type of this input is STD_LOGIC_VECTOR. Moreover, this input is 7-bit. The input called num is the binary number which is divided by 4 in the dividerBy4.vhd.
The outputs of dividerBy4.vhd
There is 1 output in dividerBy4.vhd. This output is called result which is 4-bit STD_LOGIC_VECTOR. The output called result is the resulting binary number after dividing the input called num by 8.

ONEADDERTO4BIT.VHD

OneAdderTo4Bit, using oneBitAdder as a component, calculates the sum of '1' and a 4-bit binary numbers.

The inputs of oneAdderTo4Bit.vhd
There is 1 input in oneAdderTo4Bit.vhd. The name of the input is num1 which is 4-bit STD_LOGIC_VECTOR. The input called num1 is the number which is added to 1 in oneAdderTo4Bit.vhd.
The outputs of oneAdderTo4Bit.vhd
There are 1 output in oneAdderTo4Bit.vhd. The name of the output is result which is 4-bit STD_LOGIC_VECTOR. The output called result is the number which is obtained from the addition of num1 and 1.

SUM2MEDIANS.VHD

OneAdderTo4Bit, using oneBitAdder as a component, calculates the sum of 2 4-bit binary numbers. It applies oneBitAdder to each bit one by one.

The inputs of sum2Medians.vhd
There are 2 inputs in sum2Medians.vhd. The names of the inputs are num1 and num2 which are 4-bit STD_LOGIC_VECTOR. num1 is the binary number which comes right before the middle. num2 is the binary number which comes right after the middle.
The outputs of sum2Medians.vhd
There is 1 output in sum2Medians.vhd. The name of the output is result which is 5-bit STD_LOGIC_VECTOR. The output called result is the summation of num1 and num2.

SUBTRACTOR4BIT4BIT.VHD

Subtractor4Bit4Bit, using oneBitAdder as a component, calculates the sum of complement of the second 4-bit binary number with the first 4-bit binary number. It firstly takes complement the number which is going to be subtracted, then add first number to that by using oneBitAdder component to each bit one by one.

The inputs of subtractor4bit4bit.vhd
There are 2 inputs in subtractor4bit4bit.vhd. The names of the inputs are num1 and num2 which are 4-bit STD_LOGIC_VECTOR. num2 is the binary number which is subtracted from the binary number called num1.
The outputs of subtractor4bit4bit.vhd
There is 1 output in subtractor4bit4bit.vhd. The name of the output is result which is 4-bit STD_LOGIC_VECTOR. The output called result is the outcome of the subtraction of num2 from num1.

The explanation of the taking inputs

For each number, we defined a process. Also we defined some intermediate signals called num1en, num2en, num3en, num4en, and allNumbersTaken which are all STD_LOGIC. Num1en, num2en, num3en and num4en are initialized to 1, allNumbersTaken is initialized to 0. When num1EnteredSwitch is turned on, if num1en is 1, then switch3, switch2, switch1, and switch0's values are assigned to num1's bit values from MSB to LSB respectively by process of num1, then num1en becomes 0, so process of taking first number ends. Other numbers are also taken in the same way. After the fourth number is taken, allNumbersTaken becomes 1, which means all numbers are inputted, so that calculations will start.

The explanation of the calculation of mean

By using adder4bit4bit component with port map, we summed first two numbers and got their sum, then by using adder7bit4bit component with port map, we summed the first sum with third number, and then finally again by using adder7bit4bit component with port map, we summed the second sum and fourth number. Thus, we got sum of four numbers. Then, by using dividerby4 component with port map, we divide the sum by 4, so we got the mean.

The explanation of the calculation of ranks of the numbers for calculating median and range

Before the calculation of the rank values, we have defined some intermediate signals called G12, G13, G14, G23, G24, G34 as STD_LOGIC where Gij = 1 if i>j, 0 otherwise. We calculated these values by using comparator component with port map.

Before the process definition, we have defined some intermediate signals called counter, rank1, rank2, rank3, and rank4. The type of the intermediate signal called counter is INTEGER. We have initialized the signal called counter to 1. The type of the intermediate signals called rank1, rank2, rank3, and rank4 is UNSIGNED. Each of these signals is 2-bit and each of these signals is initialized to 00. Then, we have defined a process for the calculation of the ranks of the numbers. At the beginning of the process, we checked whether all numbers are taken the counter is equal to 7, and is clock in rising edge. If counter is not equal to 7 and all numbers are taken and if rising edge exists, if the counter is equal to 1, we checked whether the G12 is equal to 1. In other words, we checked whether the num1 is greater than num2. If that is the case, we have incremented the rank1 by 1. Otherwise, we have incremented the rank2 by 1. Since the counter is equal to 1 currently, our code does not enter the other elsif statements, increments the counter by 1, and thus makes the counter equal to 2. After that, inside an elsif statement, we checked whether the counter is 2. If the counter is 2, we checked whether G13 is equal to 1. In other words, we have checked whether num1 is greater than num3. If G13 is equal to 1, then we have incremented the rank1 by 1. Otherwise, we have incremented rank3. Since the counter is currently 2, our code does not currently enter to the other elsif statements, increments the counter by 1 at the end of all if & elsif statements, and makes the counter equal to 3. Subsequently, inside an elsif statement, we have checked whether the counter is equal to 3. If the counter is 3, we have checked whether G14 is equal to 1. If G14 is equal to 1, which means if num1 is greater than num4, than we have incremented rank1 by 1. Otherwise, we have incremented rank4 by 1. We applied the same logic for the other binary combinations of the unsigned numbers, where the numbers in each of the combinations are different from each other. In other words, inside the if statments, we checked whether GAB is equal to 1. If that is the case, we have increased the rankA by 1. Otherwise, we have increased rankB by 1. To make our code enter the subsequent elsif statement after each counter value check, we have incremented the value of the counter by 1 after the end of all the if and elsif statements which do the checks of the counter values. We have done the checks for the counter values in separate elsif statements. When counter becomes 7, it means that all numbers are compared, so rank calculation ends.

***SUMMARY OF RANK CALCULATION**:* In short, after all the numbers are entered, we started a process in which each of Gij, is checked whether to be equal to 1, if it is equal to 1, rank of numi is increased by 1, otherwise rank of numj is increased by 1. We used a counter to count how many times we entered to process in order to avoid the possibility that same numbers are compared more than once.

EXPLANATION OF THE CALCULATION OF RANGE

Firstly, we have defined some intermediate signals which will be used in the code of the calculation of range. The names of these intermediate signals are max, min, and rangeminus1. The type of each of these intermediate signals is 4-bit STD_LOGIC_VECTOR. After the calculation of ranks and the definitions of the necessary intermediate signals for the calculation of range, we have defined 2 separate processes for calculating the value of range. Inside the definition of both of these processes, the parameters are rank1, rank2, rank3, rank4. The first process is for finding the maximum number in the 4-bit binary number set. Since the maximum number has a rank which is equal to 11, we checked whether each of the rank values is equal to 11 in the first process. We have done these checks in separate if & elsif statements. If rank1 is equal to 11, we have assigned the signal called max to num1. By applying the same logic to the each of the remaining ranks and binary numbers, we have found the maximum binary number in the binary number set. The second process is for finding the minimum binary number in the 4-bit binary number. The minimum number has a rank which is equal to 00, we checked whether each of the rank values is equal to 00 in the second process. We have done these checks in separate if & elsif statements. If rank1 is equal to 00, we have assigned num1 to the signal called min. By applying the same logic to the each of the remaining rank values and remaining numbers, we have found the minimum binary number in the binary number set. After the processes finish, we have used 2 port maps for calculating the range. Since max-min is equal to max+min'+1, we have first calculated the addition of max and the complement of min. Then, to calculate the value of the range, we have added 1 to the result of the addition of max and the complement of min. The first port map is the port map of the subtractor4bit4bit.vhd. The purpose of using this port map is to calculate the value of range minus 1. The second port map is the port map of the oneAdderTo4Bit.vhd. The purpose of this port map is to calculate the value of range by adding 1 to the value of range minus 1.

***SUMMARY OF RANGE CALCULATION**:* In short, we created a process which activates after each change in the ranks. This process, using if and elsif statements, finds the number with maximum rank (11) and assign max variable to it. We also created another similar process for finding min number. After finding max and min, by using subtractor4bit4bit and oneAdderTo4bit components with port map, we calculated range.

EXPLANATION OF THE CALCULATION OF MEDIAN

In the source code of the calculation of the median, we have initially defined 3 intermediate signals. The names of these intermediate signals are median1, median2, and median1PlusMedian2. The type of median1 is and median2 is 4-bit STD_LOGIC_VECTOR. The type of median1PlusMedian2 is 5-bit STD_LOGIC_VECTOR. After the definitions of the necessary intermediate signals for the calculation of median, we have defined 2 separate processes each of which has the same sensitivity list as rank1, rank2, rank3, rank4. When we write num1, num2, num3, num4 in an increasing order, we observed that the rank of the binary number right before the middle of the binary number set is equal to 01. So, we have checked whether rank1 is equal to 01 in an if statement. If that is the case, we have assigned num1 to median1. Then, in an elsif statement, we checked whether rank2 is equal to 01. If that is the case, we have assigned num2 to median1. Subsequently, again in an elsif statement, we have checked whether rank3 is equal to 01. If that is the case, we have assigned num3 to median1. By applying same logic to each of the remaining rank values and binary numbers, we have found the value of median1. median1 is the binary number which comes

right before the middle of the binary number set. In the code of the calculation of median, the purpose of the second process is to calculate the median2. median2 is the binary number which comes right after the middle of the binary number set. Moreover, the rank of the median2 is equal to 10. So, we have initially checked whether the rank1 is equal to 10 in the second process. If that is the case, we have assigned num1 to the signal called median2. After that, we have checked whether rank2 is equal to 10. If that is the case, we have assigned num2 to the signal called median2. Subsequently, we have checked whether rank3 is equal to 10. If that is the case, we have assigned num3 to the signal called median2. By applying the same logic for the remaining ranks and binary numbers, we have found the value of median2. After both of the processes end, we have used 2 port maps to calculate the median. The first port map is the port map of sum2Medians.vhd. The parameters of this port map are called median1, median2, and median1PlusMedian2. The purpose of the first port map is to sum the median1 and median2. The second port map is the port map of dividerBy2.vhd. The parameters of this port map are called median1PlusMedian2, and median. The purpose of this port map is to divide the result of the summation of median1 and median2 by 2 to calculate the median.

***SUMMARY OF MEDIAN CALCULATION**:* In short, we created a process which activates after each change in the ranks. This process, using if and elsif statements, finds the number with rank 01 and assign median1 variable to it. We also created another similar process for finding the number with rank 10. (Since middle elements have ranks 01 and 10.) After finding these middle numbers, by using sum2Medians and dividerBy2 components with port map, we calculated median.

EXPLANATION OF THE CALCULATION OF MODE

In the source code of the calculation of mode, before the processes, we have defined some intermediate signals. The names of these intermediate signals are counter2, count1, count2, count3, count4, and maxcount. The The type of counter2 is INTEGER. The intermediate signals called counter2 and counter are initialized to 1. The type of the intermediate signals called count1, count2, count3, count4 and maxcount is 2-bit UNSIGNED. We have initialized each of the unsigned data types to 00. After the definitions and initalizations of the necessary intermediate signals for the calculation of mode, we have defined 3 processes. In the first process, in an if statement, we have initially checked whether counter2 is equal to 7, whether all numbers are inputted, and whether clock is in rising edge. If counter2 is not equal to 7, all the numbers are inputted and rising edge exists, we have checked whether counter2 is equal to 1. If so, we have checked whether num1 is equal to num2. If num1 is equal to num2, we have incremented count1 and count2 by 1. Since the counter2 is currently equal to 1, our code does not currently enter the other elsif statements, increments the counter2 by 1, and thus makes the counter2 equal to 2. Subsequently, we have checked whether counter2 is equal to 2. If that is the case, inside the check of the value of counter2, we have checked whether num1 is equal to num3. If num1 is equal to num3, we have incremented count1 and count3 by 1. Until the situation where counter2 is equal to 7, by applying the same logic that is explained to counter2 and to all possible binary combinations where each of the binary numbers in the combinations are different, we have found the values of count1, count2, count3, and count4. When counter becomes 7, it means that all numbers are compared, so count calculation ends. In the second process, we have checked whether a rising edge exists. If so, inside the if statements, we have checked whether unsigned(count1) is greater than unsigned(maxcount). If so, we have assigned count1 to maxcount. Then, we have checked whether unsigned(count2) is greater than unsigned(maxcount). If the unsigned(count2) is greater than unsigned(maxcount), we have assigned count2 to maxcount. By applying the explained logic in the following elsif statements in the process, we have found the value of maxcount. In the last process, if the count1 is maximum, we have assigned num1 to mode. If the count2 is maximum, we have assigned num2 to mode. By checking whether the maxcount is equal to the each of the remaining count values, we have found the mode.

***SUMMARY OF MODE CALCULATION**:* We created a process which calculates how many times each number is repeated. For each num pair, we checked whether numi = numj, if so, we incremented counti and countj by 1. We used a counter to count how many times we entered to process in order to avoid the possibility that same numbers are compared more than once. After finding counts, with another process, we assigned the value of maxcount with comparing with each of counti. And finally, inside another process, by checking each of the counts, we found the number whose count is equal to maxcount. After finding the number with largest count, we assigned the value of mode to this number.

2. Problems encountered, errors and warnings resolved

We encountered simple syntax errors while writing our codes. Except simple syntax error, since we tried to use wait statements inside the processes, we got error and we simply resolved by this error by removing wait statements and adding clock to process. We also encountered lots of warnings during synthesis, but they are not a problem for successful synthesis of the program.

In Prometheus, we got an error about USB cable connection. We resolved this problem by using another cable. Also, we encountered some problems about the connection of FPGA board to VMWare Horizon computers since we were not familiar with usage of FPGA board. But, as we gained experience and learned about how to make VMWare recognize the USB, we finally achieved to connect FPGA board to computer properly, and make FPGA board demo successfully.

3. Conclusion

In this project, we designed a program which calculates mean, mode, median and range of inputted 4 4-bit binary numbers and displays one of these values depending on another input named modeSelect. If modeSelect=00, mean is displayed; if modeSelect=01, mode is displayed; if modeSelect=10, median is displayed; if modeSelect=11, range is displayed. In the coding, we mostly used logic gates, port maps, combinational circuit, sequential circuit etc. Therefore, during the project, we refreshed our knowledge that we gained during the lectures and especially labs. In addition, we gained a huge amount of knowledge about usage of FPGA Boards. We learned to write UCF code for programming the FPGA. Also, we learned how to use switches and other components of the board. In addition, we improved our abilities in using port map, if-else statements, and clocked processes. We understood the working principal of processes in VHDL very well.

References

1- ELEC204 Lab 1-2-3-4 Manuals
2- ELEC204 Lecture Notes
3- Youtube video for clock synchronous processes: https://www.youtube.com/watch?v=z4eqE7srNyU
4- Youtube video for process definitions: https://www.youtube.com/watch?v=VBUyqOyeueI

Appendix 1. Lab source code
Appendix 2. RTL schematics
Appendix 3. Screenshots from Xilinx for the errors and other board issues
Appendix 4. FPGA Board photos showing working code
Appendix 5. The screenshots and the explanations of the simulation results

Appendix 1. Lab source code

Source code of Main.vhd

```
library IEEE;
use ieee.numeric_std.all;
use IEEE.STD_LOGIC_1164.ALL;

entity Main is
   Port ( --clock for processes
                                    clk : in STD_LOGIC;

                                    -- switchs for inputting numbers
                                    switch0: in STD_LOGIC;
                                    switch1: in STD_LOGIC;
                                    switch2: in STD_LOGIC;
                                    switch3: in STD_LOGIC;

                                    -- switchs for assigning value of modeSelect
                                    switch8: in STD_LOGIC;
                                    switch9: in STD_LOGIC;

                                    --switchs for saving each number
                                    num1EnteredSwitch: in STD_LOGIC;
                                    num2EnteredSwitch: in STD_LOGIC;
                                    num3EnteredSwitch: in STD_LOGIC;
                                    num4EnteredSwitch: in STD_LOGIC;

                                    -- output
                                    result : out  STD_LOGIC_VECTOR (3 downto 0)
                                    );

end Main;

architecture Behavioral of Main is

        --to sum 2 4-bit numbers
        component adder4bit4bit
                Port ( cin : in  STD_LOGIC;
                                    num1 : in  STD_LOGIC_VECTOR (3 downto 0);
      num2 : in  STD_LOGIC_VECTOR (3 downto 0);
                                    result : out  STD_LOGIC_VECTOR (6 downto 0);
      Cout : out  STD_LOGIC);
        end component;

        --to sum 7-bit 4-bit numbers
        component adder7bit4bit
                Port ( cin : in  STD_LOGIC;
                                    num1 : in  STD_LOGIC_VECTOR (6 downto 0);
      num2 : in  STD_LOGIC_VECTOR (3 downto 0);
                                    result : out  STD_LOGIC_VECTOR (6 downto 0);
      Cout : out  STD_LOGIC);
        end component;


        component dividerBy4
                Port (
                                    num : in  STD_LOGIC_VECTOR (6 downto 0);
      result : out  STD_LOGIC_VECTOR (3 downto 0));
        end component;
```

```
		component dividerBy2
			Port (
				num : in  STD_LOGIC_VECTOR (4 downto 0);
	result : out  STD_LOGIC_VECTOR (3 downto 0));
		end component;


		component comparator
Port ( A : in  STD_LOGIC_VECTOR (3 DOWNTO 0);
	B : in  STD_LOGIC_VECTOR (3 DOWNTO 0);
	GAB : out  STD_LOGIC);
		end component;


		component subtractor4bit4bit
			Port (
				num1 : in  STD_LOGIC_VECTOR (3 downto 0);
	num2 : in  STD_LOGIC_VECTOR (3 downto 0);
				result : out  STD_LOGIC_VECTOR (3 downto 0));
		end component;

		--to add 1 to 4-bit number
		component oneAdderTo4Bit
		  Port (
		  num1 : in  STD_LOGIC_VECTOR (3 DOWNTO 0);
 result : out  STD_LOGIC_VECTOR (3 DOWNTO 0)
		  );
		end component;

		component sum2Medians
			Port (
				num1 : in  STD_LOGIC_VECTOR (3 downto 0);
	num2 : in  STD_LOGIC_VECTOR (3 downto 0);
				result : out  STD_LOGIC_VECTOR (4 downto 0)
	);
		end component;


		-- numbers
		SIGNAL num1 : STD_LOGIC_VECTOR (3 downto 0);
		SIGNAL num2 : STD_LOGIC_VECTOR (3 downto 0);
		SIGNAL num3 : STD_LOGIC_VECTOR (3 downto 0);
		SIGNAL num4 : STD_LOGIC_VECTOR (3 downto 0);

		-- signal which decide to operation (00=mean, 01=mode, 10=median, 11=range)
		SIGNAL modeSelect : STD_LOGIC_VECTOR(1 downto 0);

		-- results of calculations
		SIGNAL mean : STD_LOGIC_VECTOR (3 downto 0);
		SIGNAL mode :STD_LOGIC_VECTOR (3 downto 0);
		SIGNAL median: STD_LOGIC_VECTOR(3 downto 0);
		SIGNAL rangee :  STD_LOGIC_VECTOR (3 downto 0);

		-- sums which are used for mean
		SIGNAL sum1 : STD_LOGIC_VECTOR (6 downto 0):="0000000";
		SIGNAL sum2 : STD_LOGIC_VECTOR (6 downto 0):="0000000";
		SIGNAL sum3 : STD_LOGIC_VECTOR (6 downto 0):="0000000";

		-- result of comparisons (if Gij=1, it means i>j)
		SIGNAL G12 : STD_LOGIC;
		SIGNAL G13 : STD_LOGIC;
		SIGNAL G14 : STD_LOGIC;
		SIGNAL G23 : STD_LOGIC;
		SIGNAL G24 : STD_LOGIC;
```

```
SIGNAL G34 : STD_LOGIC;

--for range
SIGNAL max : STD_LOGIC_VECTOR (3 downto 0):="0000";
SIGNAL min : STD_LOGIC_VECTOR (3 downto 0):="0000";
SIGNAL cout : STD_LOGIC_VECTOR(3 downto 1);
signal rangeminus1 : STD_LOGIC_VECTOR (3 downto 0);

--for median
SIGNAL median1 : STD_LOGIC_VECTOR (3 downto 0);
SIGNAL median2 : STD_LOGIC_VECTOR (3 downto 0);
SIGNAL median1PlusMedian2 : STD_LOGIC_VECTOR (4 downto 0);

--counters to use in processes which calculate ranks and counts.
SIGNAL counter : INTEGER := 1;
SIGNAL counter2 : INTEGER := 1;

-- ranks of the numbers (number with rank 11 is biggest, number with rank 00 is smallest.)
SIGNAL rank1 : UNSIGNED (1 downto 0) := "00";
SIGNAL rank2 : UNSIGNED (1 downto 0) := "00";
SIGNAL rank3 : UNSIGNED (1 downto 0) := "00";
SIGNAL rank4 : UNSIGNED (1 downto 0) := "00";

-- counts (how many times repeated) of the numbers (used for mode)
SIGNAL count1 : UNSIGNED (1 downto 0) := "00";
SIGNAL count2 : UNSIGNED (1 downto 0) := "00";
SIGNAL count3 : UNSIGNED (1 downto 0) := "00";
SIGNAL count4 : UNSIGNED (1 downto 0) := "00";
SIGNAL maxcount : UNSIGNED (1 downto 0) := "00";

--enable inputs for taking numbers from switchs.
SIGNAL num1en: STD_LOGIC :='1';
SIGNAL num2en: STD_LOGIC :='1';
SIGNAL num3en :STD_LOGIC :='1';
SIGNAL num4en :STD_LOGIC :='1';
SIGNAL allNumbersTaken : STD_LOGIC :='0';


begin


modeSelect(0)<=switch8;
modeSelect(1)<=switch9;

--sum each number one by one
summing1:  adder4bit4bit  port map (  '0'  , num1, num2, sum1, cout(1));
summing2:  adder7bit4bit port map (cout(1), sum1, num3, sum2, cout(2));
summing3:  adder7bit4bit port map (cout(2), sum2, num4, sum3, cout(3));

--divide by 4
calculateMean: dividerBy4 port map (sum3, mean);
----mean ends----

--compare numbers (used for rank calculations)
calculateG12: comparator port map (num1, num2, G12);
calculateG13: comparator port map (num1, num3, G13);
calculateG14: comparator port map (num1, num4, G14);

calculateG23: comparator port map (num2, num3, G23);
calculateG24: comparator port map (num2, num4, G24);

calculateG34: comparator port map (num3, num4, G34);


--processes which take the number inputs from switches and assign them to num1, num2, num3 and num4.
process (num1EnteredSwitch)
```

```
	begin

				if(num1en='1' and num1EnteredSwitch='1') then
					num1(0)<=switch0;
					num1(1)<=switch1;
					num1(2)<=switch2;
					num1(3)<=switch3;
					num1en<='0';
				end if;

	end process;

	process (num2EnteredSwitch)
	begin
				if(num2en='1' and num2EnteredSwitch='1') then
					num2(0)<=switch0;
					num2(1)<=switch1;
					num2(2)<=switch2;
					num2(3)<=switch3;
					num2en<='0';
				end if;

	end process;

	process (num3EnteredSwitch)
	begin
				if(num3en='1' and num3EnteredSwitch='1') then
					num3(0)<=switch0;
					num3(1)<=switch1;
					num3(2)<=switch2;
					num3(3)<=switch3;
					num3en<='0';
				end if;
	end process;

	process (num4EnteredSwitch)
	begin
				if(num4en='1' and num4EnteredSwitch='1') then
					num4(0)<=switch0;
					num4(1)<=switch1;
					num4(2)<=switch2;
					num4(3)<=switch3;
					num4en<='0';
					allNumbersTaken<='1';
				end if;
	end process;


	--ranks are calculated to use for median and range
	process (clk, num4)
	begin

if (rising_edge(clk) and counter /= 7 and allNumbersTaken='1') then


				if(counter = 1) then
					if (G12 = '1') then
						rank1 <= rank1 + 1;
					else
						rank2 <= rank2 + 1;
					end if;
				elsif (counter = 2) then
					if (G13 = '1') then
						rank1 <= rank1 + 1;
					else
						rank3 <= rank3 + 1;
```

```
                        end if;
                elsif (counter = 3) then
                        if (G14 = '1') then
                                rank1 <= rank1 + 1;
                        else
                                rank4 <= rank4 + 1;
                        end if;
                elsif (counter = 4) then
                        if (G23 = '1') then
                                rank2 <= rank2 + 1;
                        else
                                rank3 <= rank3 + 1;
                        end if;
                elsif (counter = 5) then
                        if (G24 = '1') then
                                rank2 <= rank2 + 1;
                        else
                                rank4 <= rank4 + 1;
                        end if;

                elsif (counter = 6) then
                        if (G34 = '1') then
                                rank3 <= rank3 + 1;
                        else
                                rank4 <= rank4 + 1;
                        end if;

                end if;
        counter <= counter + 1 ;
end if;
end process;


----range starts----

--max num is found.
process (rank1,rank2,rank3,rank4)
begin
        if(rank1 = "11") then
                max <= num1;
        elsif (rank2 = "11") then
                max <= num2;
        elsif (rank3 = "11") then
                max <= num3;
        elsif (rank4 = "11") then
                max <= num4;
        end if;
end process;

--min num is found.
process (rank1,rank2,rank3,rank4)
begin
        if(rank1 = "00") then
                min <= num1;
        elsif (rank2 = "00") then
                min <= num2;
        elsif (rank3 = "00") then
                min <= num3;
        elsif (rank4 = "00") then
                min <= num4;
        end if;
end process;

--subtract min from max (first compute A+B', then add 1 since A-B = A+B'-1)
calculateRangeMinus1: subtractor4bit4bit  port map (max,  min, rangeminus1);
calculateRange: oneAdderTo4Bit  port map (rangeminus1, rangee);
```

```
	----range ends----

	----median starts----

	--find the number with rank 1 and 2 which are middle numbers.
	process (rank1,rank2,rank3,rank4)
	begin
		if(rank1 = "01") then
			median1 <= num1;
		elsif (rank2 = "01") then
			median1 <= num2;
		elsif (rank3 = "01") then
			median1 <= num3;
		elsif (rank4 = "01") then
			median1 <= num4;
		end if;
	end process;

	process (rank1,rank2,rank3,rank4)
	begin
		if(rank1 = "10") then
			median2 <= num1;
		elsif (rank2 = "10") then
			median2 <= num2;
		elsif (rank3 = "10") then
			median2 <= num3;
		elsif (rank4 = "10") then
			median2 <= num4;
		end if;
	end process;

	--sum two middle numbers and divide by two to get median.
	sumMedian1andMedian2: sum2Medians port map (median1, median2, median1PlusMedian2);
	calculateMedian: dividerBy2 port map (median1PlusMedian2, median);
----median ends----

	----mode starts----

	--calculate count of numbers.
	process (clk, num4)
	begin

if (rising_edge(clk) and counter2 /= 7 and allNumbersTaken='1') then

				if(counter2 = 1) then
					if (num1 = num2 ) then
						count1 <= count1 + 1;
						count2 <= count2 + 1;
					end if;
				elsif (counter2 = 2) then
					if (num1 = num3 ) then
						count1 <= count1 + 1;
						count3 <= count3 + 1;
					end if;
				elsif (counter2 = 3) then
					if (num1 = num4 ) then
						count1 <= count1 + 1;
						count4 <= count4 + 1;
					end if;

				elsif (counter2 = 4) then
					if (num2 =num3 ) then
						count2 <= count2 + 1;
						count3 <= count3 + 1;
					end if;
				elsif (counter2 = 5) then
```

```
                                        if (num2 = num4 ) then
                                                count2 <= count2 + 1;
                                                count4 <= count4 + 1;
                                        end if;

                                elsif (counter2 = 6) then
                                        if (num3 = num4) then
                                                count3 <= count3 + 1;
                                                count4 <= count4 + 1;
                                        end if;

                                end if;
                        counter2 <= counter2 + 1 ;
                end if;
                end process;

                --find the largest count
                process (clk, count1, count2, count3, count4)
                begin

                        if(rising_edge(clk)) then

                                        if (unsigned(count1)>unsigned(maxcount)) then
                                                maxcount<=count1;
                                        end if;

                                        if (unsigned(count2)>unsigned(maxcount)) then
                                                maxcount<=count2;
                                        end if;

                                        if (unsigned(count3)>unsigned(maxcount)) then
                                                maxcount<=count3;
                                        end if;

                                        if (unsigned(count4)>unsigned(maxcount)) then
                                                maxcount<=count4;
                                        end if;

                end if;
                end process;

                --assign the value of mode to number which has the largest count.
                process ( maxcount,count1,count2,count3,count4)
                begin
                        if(maxcount = count1) then
                                mode <= num1;
                        elsif (maxcount = count2) then
                                mode <= num2;
                        elsif (maxcount = count3) then
                                mode <= num3;
                        elsif (maxcount = count4) then
                                mode <= num4;
                        end if;
                end process;
                ----mode ends----

                --assign result's value depending on the value of modeSelect.
                result <=  mean when modeSelect = "00" else
                                        mode when modeSelect = "01" else
                                        median when modeSelect = "10" else
                                        rangee;

end Behavioral;
```

The source code of comparator.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity comparator is
   Port ( A : in  STD_LOGIC_VECTOR (3 DOWNTO 0);
          B : in  STD_LOGIC_VECTOR (3 DOWNTO 0);
          GAB : out  STD_LOGIC);
end comparator;

architecture Behavioral of comparator is


               SIGNAL a3eqb3: STD_LOGIC;
               SIGNAL a2eqb2: STD_LOGIC;
               SIGNAL a1eqb1: STD_LOGIC;
               SIGNAL a0eqb0: STD_LOGIC;

begin


               a3eqb3 <= A(3) xnor B(3);
               a2eqb2 <= A(2) xnor B(2);
               a1eqb1 <= A(1) xnor B(1);
               a0eqb0 <= A(0) xnor B(0);



               GAB <= (A(3) and not B(3))
                        or (a3eqb3 and (a(2) and not b(2)))
                        or (a3eqb3 and a2eqb2 and (a(1) and not b(1)))
                       or (a3eqb3 and a2eqb2 and a1eqb1 and (a(0) and not b(0)));


end Behavioral;
```

The source code of adder4bit4bit.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity adder4bit4bit is
               Port (
               cin : in STD_LOGIC;
               num1 : in  STD_LOGIC_VECTOR (3 DOWNTO 0);
               num2 : in  STD_LOGIC_VECTOR (3 DOWNTO 0);
    result : out  STD_LOGIC_VECTOR (6 DOWNTO 0);
               Cout : out  STD_LOGIC);

end adder4bit4bit;

architecture Behavioral of adder4bit4bit is
               component oneBitAdder
                              Port ( oneBitCin : in  STD_LOGIC;
          oneBitNum1 : in  STD_LOGIC;
          oneBitNum2 : in  STD_LOGIC;
          oneBitResult : out  STD_LOGIC;
          oneBitCout : out  STD_LOGIC);
               end component;

               SIGNAL C : STD_LOGIC_VECTOR (11 DOWNTO 1);
begin

               zerothBit:  oneBitAdder port map (cin ,num1(0),num2(0),result(0),C(1));
```

```
        firstBit:   oneBitAdder port map (C(1),num1(1),num2(1),result(1),C(2));
        secondBit:  oneBitAdder port map (C(2),num1(2),num2(2),result(2),C(3));
        thirdBit:   oneBitAdder port map (C(3),num1(3),num2(3),result(3),C(4));
        forthBit:   oneBitAdder port map (C(4),'0','0',result(4),C(5));
        fifthBit:   oneBitAdder port map (C(5),'0','0',result(5),C(6));
        sixthBit:   oneBitAdder port map (C(6),'0','0',result(6),C(7));

end Behavioral;
```

Source code of adder7bit4bit.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity adder7bit4bit is
        Port (
        cin : in STD_LOGIC;
        num1 : in  STD_LOGIC_VECTOR (6 DOWNTO 0);
        num2 : in  STD_LOGIC_VECTOR (3 DOWNTO 0);
   result : out  STD_LOGIC_VECTOR (6 DOWNTO 0);
        Cout : out  STD_LOGIC);

end adder7bit4bit;

architecture Behavioral of adder7bit4bit is
        component oneBitAdder
                Port ( oneBitCin : in  STD_LOGIC;
                    oneBitNum1 : in  STD_LOGIC;
                oneBitNum2 : in  STD_LOGIC;
                 oneBitResult : out  STD_LOGIC;
             oneBitCout : out  STD_LOGIC);
         end component;

        SIGNAL C : STD_LOGIC_VECTOR (7 DOWNTO 1);

begin

        zerothBit:  oneBitAdder port map (cin ,num1(0),num2(0),result(0),C(1));
        firstBit:   oneBitAdder port map (C(1),num1(1),num2(1),result(1),C(2));
        secondBit:  oneBitAdder port map (C(2),num1(2),num2(2),result(2),C(3));
        thirdBit:   oneBitAdder port map (C(3),num1(3),num2(3),result(3),C(4));
        forthBit:   oneBitAdder port map (C(4),num1(4),'0',result(4),C(5));
        fifthBit:   oneBitAdder port map (C(5),num1(5),'0',result(5),C(6));
        sixthBit:   oneBitAdder port map (C(6),num1(6),'0',result(6),C(7));


 end Behavioral;
```

Source code of dividerBy2.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity dividerBy2 is
   Port ( num : in  STD_LOGIC_VECTOR(4 downto 0);
       result : out  STD_LOGIC_VECTOR(3 downto 0));
end dividerBy2;

architecture Behavioral of dividerBy2 is

begin
        result(0) <= num(1);
        result(1) <= num(2);
```

```
        result(2) <= num(3);
        result(3) <= num(4);

end Behavioral;
```

Source code of dividerBy4.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity dividerBy4 is
   Port ( num : in  STD_LOGIC_VECTOR(6 downto 0);
          result : out  STD_LOGIC_VECTOR(3 downto 0));
end dividerBy4;

architecture Behavioral of dividerBy4 is

begin
        result(0) <= num(2);
        result(1) <= num(3);
        result(2) <= num(4)
        result(3) <= num(5);

end Behavioral;
```

Source code of oneAdderTo4Bit.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity oneAdderTo4Bit is
          Port (
          num1 : in  STD_LOGIC_VECTOR (3 DOWNTO 0);
    result : out  STD_LOGIC_VECTOR (3 DOWNTO 0)
          );

end oneAdderTo4Bit;

architecture Behavioral of oneAdderTo4Bit is
        component oneBitAdder
                Port ( oneBitCin : in  STD_LOGIC;
                   oneBitNum1 : in  STD_LOGIC;
          oneBitNum2 : in  STD_LOGIC;
                                oneBitResult : out  STD_LOGIC;
          oneBitCout : out  STD_LOGIC);
        end component;

        SIGNAL C : STD_LOGIC_VECTOR (4 DOWNTO 1);
begin

        zerothBit:  oneBitAdder port map ('0' ,num1(0),'1',result(0),C(1));
        firstBit:   oneBitAdder port map (C(1),num1(1),'0',result(1),C(2));
        secondBit:  oneBitAdder port map (C(2),num1(2),'0',result(2),C(3));
        thirdBit:   oneBitAdder port map (C(3),num1(3),'0',result(3),C(4));


end Behavioral;
```

Source code of oneBitAdder.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity oneBitAdder is
   Port ( oneBitCin : in  STD_LOGIC;
                  oneBitNum1 : in  STD_LOGIC;
        oneBitNum2 : in  STD_LOGIC;
                                    oneBitResult : out  STD_LOGIC;
        oneBitCout : out  STD_LOGIC);
end oneBitAdder;

architecture Behavioral of oneBitAdder is
              SIGNAL num1ToSum:STD_LOGIC;
              SIGNAL num2ToSum:STD_LOGIC;
              SIGNAL cToSum:STD_LOGIC;


begin
                        num1ToSum<= '0' when oneBitNum1='U' else
                                                              oneBitNum1;


                        num2ToSum<= '0' when oneBitNum2='U' else
                                                              oneBitNum2;


                        cToSum<= '0' when oneBitCin='U' else
                                                          oneBitCin;

                        oneBitResult <= (num1ToSum xor num2ToSum) xor cToSum;
                        oneBitCout <=  (num1ToSum and num2ToSum) or (num1ToSum and cToSum) or (num2ToSum and cToSum);

end Behavioral;
```

Source code of sum2Medians.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity sum2Medians is
               Port (
               num1 : in  STD_LOGIC_VECTOR (3 DOWNTO 0);
               num2 : in  STD_LOGIC_VECTOR (3 DOWNTO 0);
    result : out  STD_LOGIC_VECTOR (4 DOWNTO 0)
               );

end sum2Medians;

architecture Behavioral of sum2Medians is
              component oneBitAdder
                        Port ( oneBitCin : in  STD_LOGIC;
                           oneBitNum1 : in  STD_LOGIC;
        oneBitNum2 : in  STD_LOGIC;
                                    oneBitResult : out  STD_LOGIC;
        oneBitCout : out  STD_LOGIC);
              end component;

              SIGNAL C : STD_LOGIC_VECTOR (11 DOWNTO 1);
begin

              zerothBit:  oneBitAdder port map ('0' ,num1(0),num2(0),result(0),C(1));
              firstBit:   oneBitAdder port map (C(1),num1(1),num2(1),result(1),C(2));
              secondBit:  oneBitAdder port map (C(2),num1(2),num2(2),result(2),C(3));
              thirdBit:   oneBitAdder port map (C(3),num1(3),num2(3),result(3),C(4));
              forthBit:   oneBitAdder port map (C(4),'0','0',result(4),C(5));


end Behavioral;
```

Source code of subtractor4bit4bit.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity subtractor4bit4bit is
                Port (
                num1 : in  STD_LOGIC_VECTOR (3 DOWNTO 0);
                num2 : in  STD_LOGIC_VECTOR (3 DOWNTO 0);
    result : out  STD_LOGIC_VECTOR (3 DOWNTO 0)
                );

end subtractor4bit4bit;

architecture Behavioral of subtractor4bit4bit is
                component oneBitAdder
                                Port ( oneBitCin : in  STD_LOGIC;
                                   oneBitNum1 : in  STD_LOGIC;
        oneBitNum2 : in  STD_LOGIC;
                                                oneBitResult : out  STD_LOGIC;
        oneBitCout : out  STD_LOGIC);
                end component;

                SIGNAL C : STD_LOGIC_VECTOR (4 DOWNTO 1);
                SIGNAL complementOfNum2: STD_LOGIC_VECTOR (3 DOWNTO 0);
begin

                complementOfNum2 <= not num2;
                zerothBit:  oneBitAdder port map ('0' ,num1(0),complementOfNum2(0),result(0),C(1));
                firstBit:   oneBitAdder port map (C(1),num1(1),complementOfNum2(1),result(1),C(2));
                secondBit:  oneBitAdder port map (C(2),num1(2),complementOfNum2(2),result(2),C(3));
                thirdBit:   oneBitAdder port map (C(3),num1(3),complementOfNum2(3),result(3),C(4));

end Behavioral;
```

Simulation Code (fpgaTest4Numbers.vhd)

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
USE ieee.numeric_std.ALL;

ENTITY fpgaTest4numbers IS
END fpgaTest4numbers;

ARCHITECTURE behavior OF fpgaTest4numbers IS

   -- Component Declaration for the Unit Under Test (UUT)

   COMPONENT Main
   PORT(
      clk : IN  std_logic;
      switch0 : IN  std_logic;
      switch1 : IN  std_logic;
      switch2 : IN  std_logic;
      switch3 : IN  std_logic;
      switch8 : IN  std_logic;
      switch9 : IN  std_logic;
      num1EnteredSwitch : IN  std_logic;
      num2EnteredSwitch : IN  std_logic;
      num3EnteredSwitch : IN  std_logic;
      num4EnteredSwitch : IN  std_logic;
```

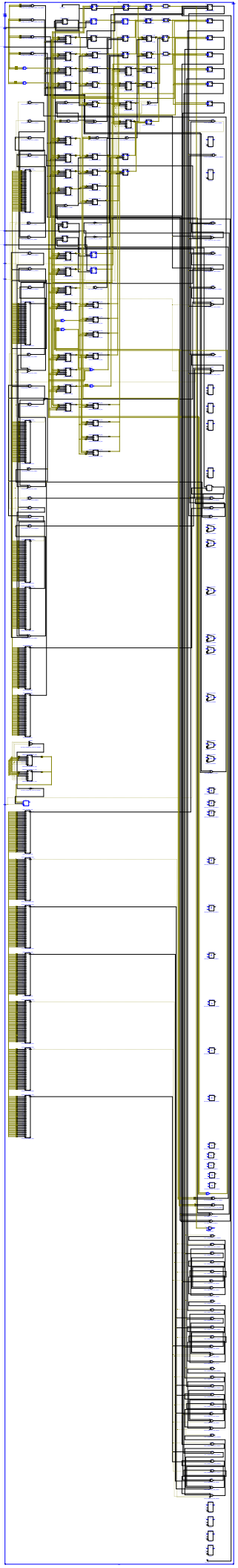
```
      result : OUT  std_logic_vector(3 downto 0)
     );
 END COMPONENT;


  --Inputs
  signal clk : std_logic := '0';
  signal switch0 : std_logic := '0';
  signal switch1 : std_logic := '0';
  signal switch2 : std_logic := '0';
  signal switch3 : std_logic := '0';
  signal switch8 : std_logic := '0';
  signal switch9 : std_logic := '0';
  signal num1EnteredSwitch : std_logic := '0';
  signal num2EnteredSwitch : std_logic := '0';
  signal num3EnteredSwitch : std_logic := '0';
  signal num4EnteredSwitch : std_logic := '0';

        --Outputs
  signal result : std_logic_vector(3 downto 0);

  -- Clock period definitions
  constant clk_period : time := 1 ns;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
  uut: Main PORT MAP (
       clk => clk,
       switch0 => switch0,
       switch1 => switch1,
       switch2 => switch2,
       switch3 => switch3,
       switch8 => switch8,
       switch9 => switch9,
       num1EnteredSwitch => num1EnteredSwitch,
       num2EnteredSwitch => num2EnteredSwitch,
       num3EnteredSwitch => num3EnteredSwitch,
       num4EnteredSwitch => num4EnteredSwitch,
       result => result
      );

  -- Clock process definitions
  clk_process :process
  begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
  end process;


  -- Stimulus process
  stim_proc: process
  begin
    -- hold reset state for 100 ns.
    wait for 10 ns;
                --num1 = 0010  =  2
                switch3<='0';wait for 10 ns;switch2<='0';wait for 10 ns;switch1<='1';wait for 10 ns;switch0<='0';wait for 10 ns;
                num1EnteredSwitch<='1';wait for 10 ns;
                --num2 = 1110  =  14
                switch3<='1';wait for 10 ns;switch2<='1';wait for 10 ns;switch1<='1';wait for 10 ns;switch0<='0';wait for 10 ns;
                num2EnteredSwitch<='1';wait for 10 ns;
                --num3 = 0010  =  2
                switch3<='0';wait for 10 ns;switch2<='0';wait for 10 ns;switch1<='1';wait for 10 ns;switch0<='0';wait for 10 ns;
                num3EnteredSwitch<='1';wait for 10 ns;
```

```
                --num4 = 1010  =  10
                switch3<='1';wait for 10 ns;switch2<='0';wait for 10 ns;switch1<='1';wait for 10 ns;switch0<='0';wait for 10 ns;
                num4EnteredSwitch<='1';wait for 10 ns;

                --{2,14,2,10} -> mean=7, mode=2, median=6, range=12

                                                --initially modeselect=00 -> mean displayed
                switch9<='0';switch8<='1';wait for 10 ns; --modeselect=01 -> mode displayed
                switch9<='1';switch8<='0';wait for 10 ns; --modeselect=10 -> median displayed
                switch9<='1';switch8<='1';wait for 10 ns; --modeSelect=11 -> range displayed


    wait;
  end process;

END;
```

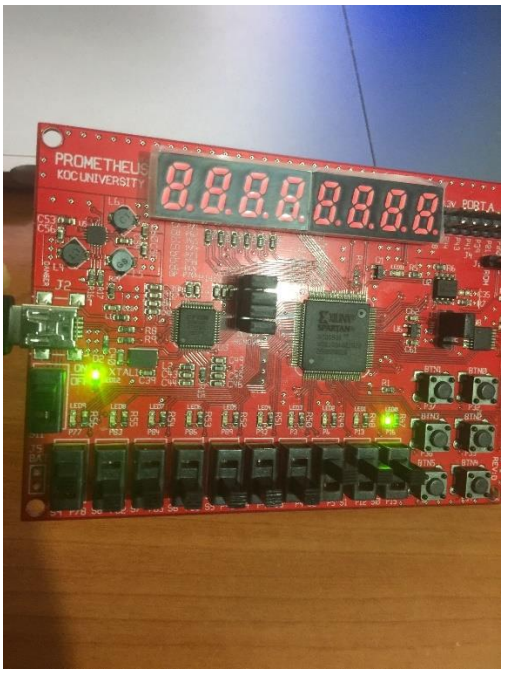
Pins4Numbers.ucf

```
NET "SWITCH0" LOC = "P15";
NET "SWITCH1" LOC = "P12";
NET "SWITCH2" LOC = "P5";
NET "SWITCH3" LOC = "P4";
NET "SWITCH8" LOC = "P94";
NET "SWITCH9" LOC = "P90";
NET "NUM1ENTEREDSWITCH" LOC = "P78";
NET "NUM2ENTEREDSWITCH" LOC = "P82";
NET "NUM3ENTEREDSWITCH" LOC = "P85";
NET "NUM4ENTEREDSWITCH" LOC = "P88";
NET "RESULT(3)" LOC = "P3";
NET "RESULT(2)" LOC = "P6";
NET "RESULT(1)" LOC = "P13";
NET "RESULT(0)" LOC = "P16";
NET "CLK" LOC = "P40";
NET "num1EnteredSwitch" CLOCK_DEDICATED_ROUTE = FALSE;
NET "num2EnteredSwitch" CLOCK_DEDICATED_ROUTE = FALSE;
NET "num3EnteredSwitch" CLOCK_DEDICATED_ROUTE = FALSE;
NET "num4EnteredSwitch" CLOCK_DEDICATED_ROUTE = FALSE;
```

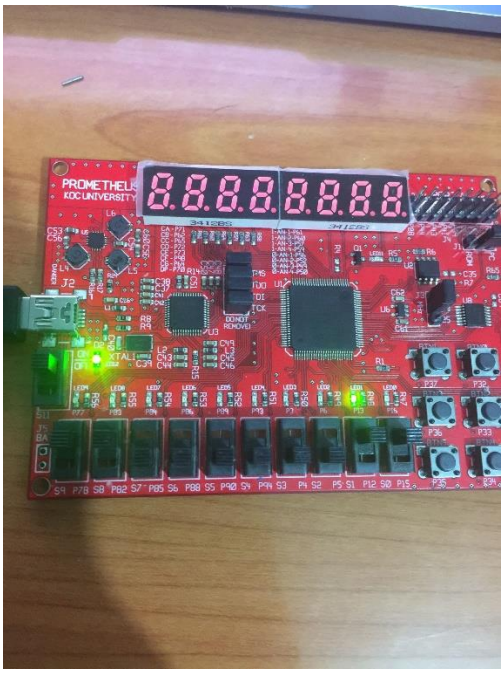
Appendix 2. RTL schematics *(in next page)*



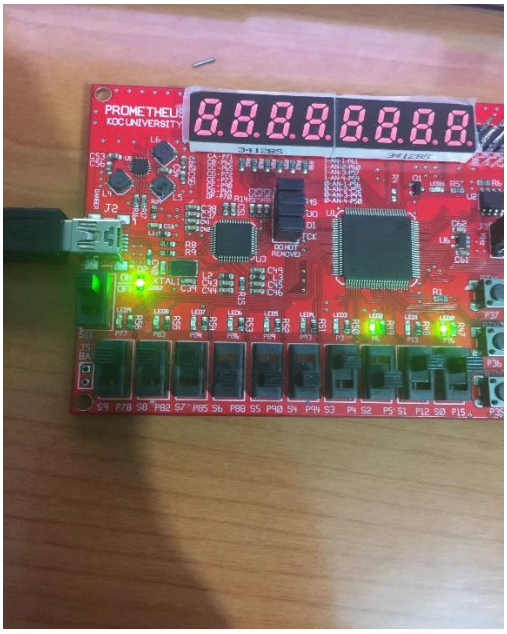
Appendix 3. FPGA Board photos showing working code



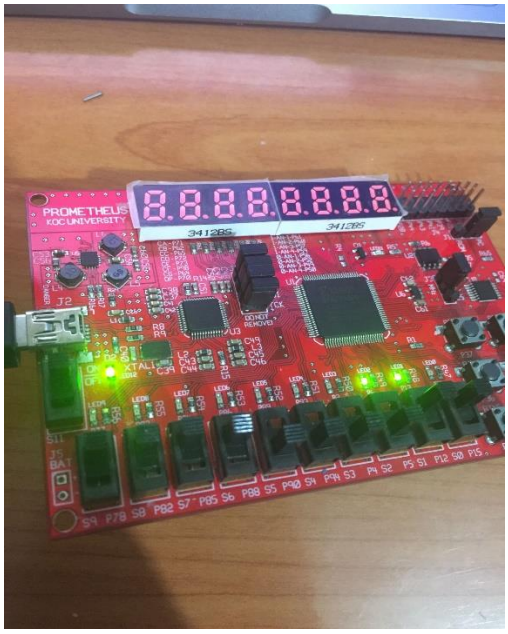
Num 1 is entered and saved



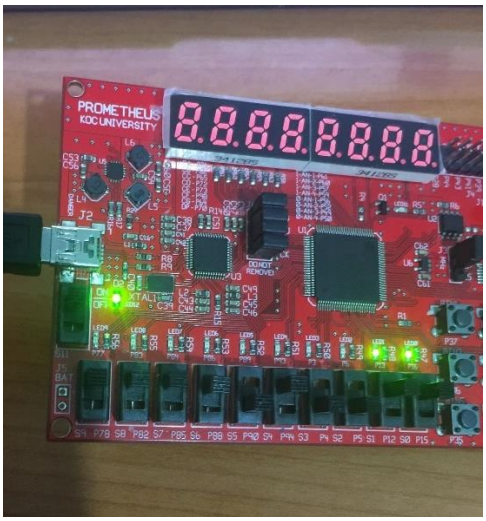
Num 2 is entered and saved



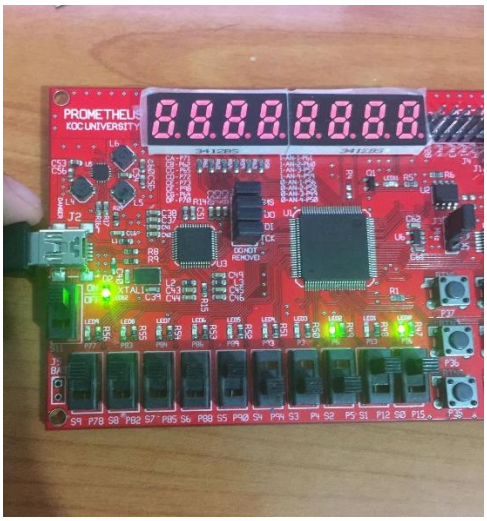
Num 3 is entered and saved



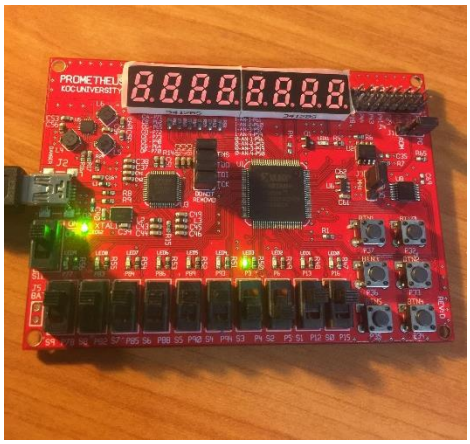
Num 4 is entered and saved, and also since modeSelect is 00, mean (0110) is displayed in LEDs.



Since modeSelect=01, mode (0011) is displayed in LEDS.



Since modeSelect=10, median (0101) is displayed in LEDS.



Since modeSelect=11, range (1000) is displayed in LEDs.

SUMMARY OF FPGA BOARD PHOTOS:

INPUTTED NUMBERS:

num1: 0111 (7 in decimal)
num2: 0011 (3 in decimal)
num3: 1011 (11 in decimal)
num4: 0011 (3 in decimal)

- ⇨ Mean = 24/4 = 6 (0110 in binary)
- ⇨ Mode = 3 (0011 in binary)
- ⇨ Median = (3+7)/2 = 5 (0101 in binary)
- ⇨ Range = 11-3=8 (1000 in binary)

NOTE: Results are displayed in Led3, Led2, Led1 and Led0. For example, if result is 0110, Led2 and Led1 give light.

Appendix 4. Screenshots from Xilinx for the errors and other board issues

Console

```
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/Users/iokay/project4bit4number/main.vhd" into library work
ERROR:HDLCompiler:806 - "C:/Users/iokay/project4bit4number/main.vhd" Line 450: Syntax error near "elsif".
ERROR:ProjectMgmt - 1 error(s) found while parsing design hierarchy.
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/Users/iokay/project4bit4number/main.vhd" into library work
ERROR:HDLCompiler:806 - "C:/Users/iokay/project4bit4number/main.vhd" Line 450: Syntax error near "elsif".
ERROR:ProjectMgmt - 1 error(s) found while parsing design hierarchy.
```

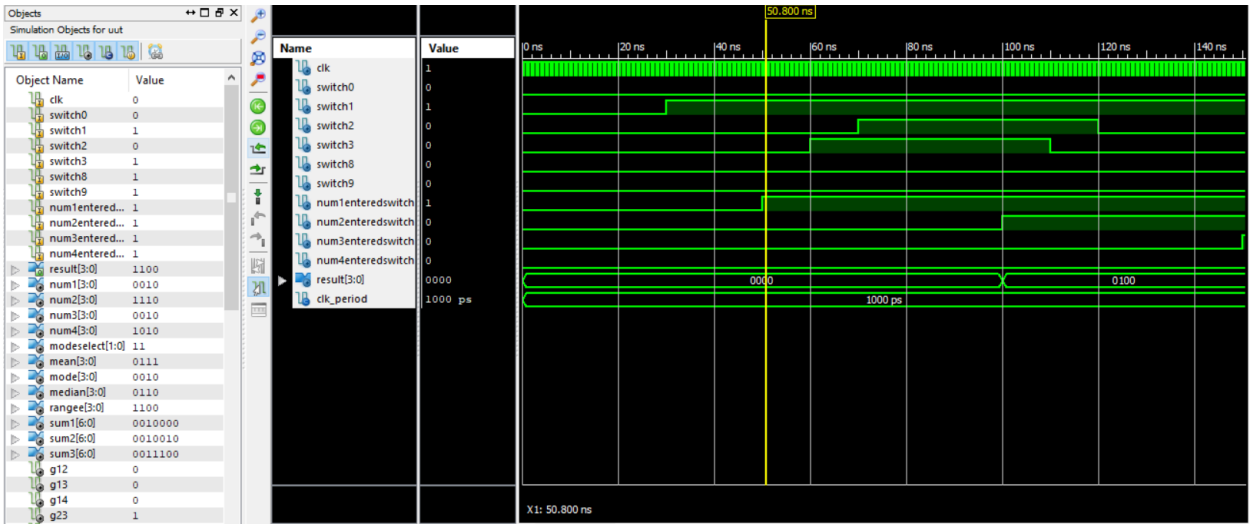
Simple syntax error

Error caused by using wait statement inside process

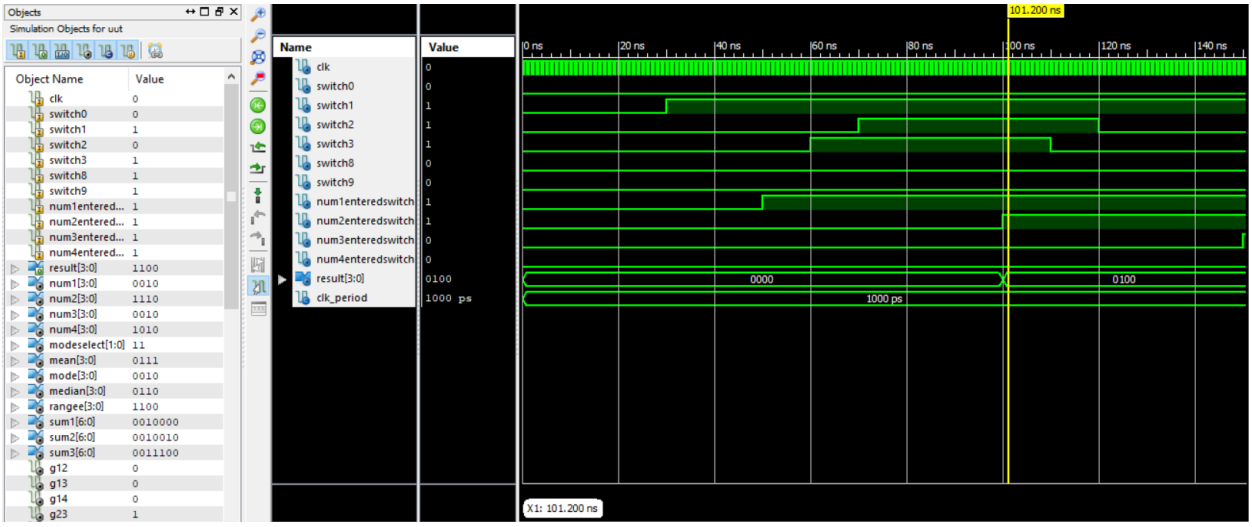
Prometheus USB connection error

Appendix 5. The screenshots and the explanations of the simulation results

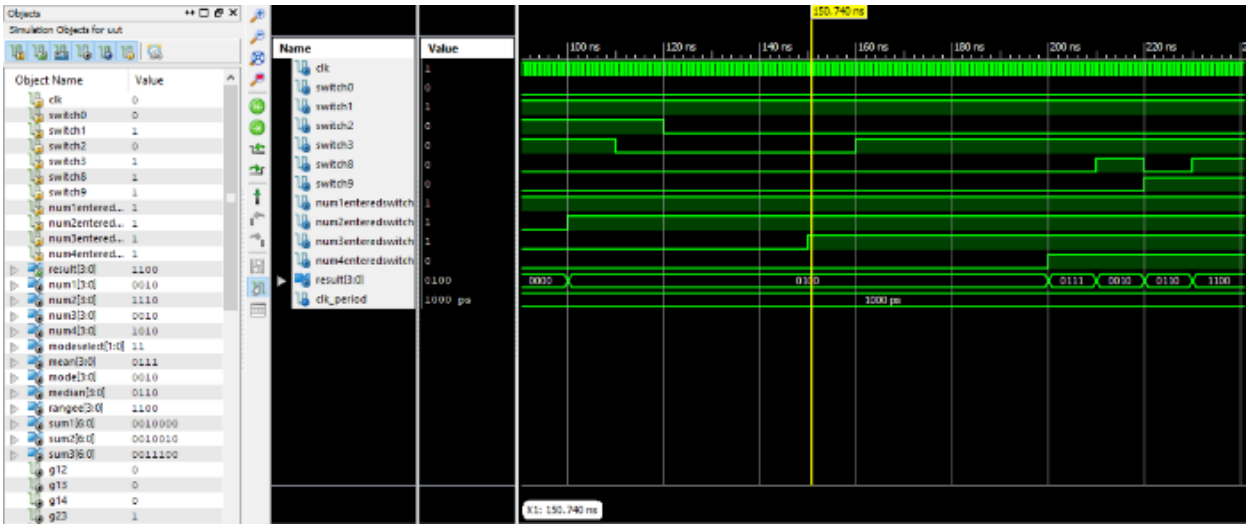
Since num1enteredswitch is turned on (became 1), the bits of num1 are assigned to the value switch3, switch2, switch1 and switch0, respectively. Therefore, num1=0010 (2 in decimal).



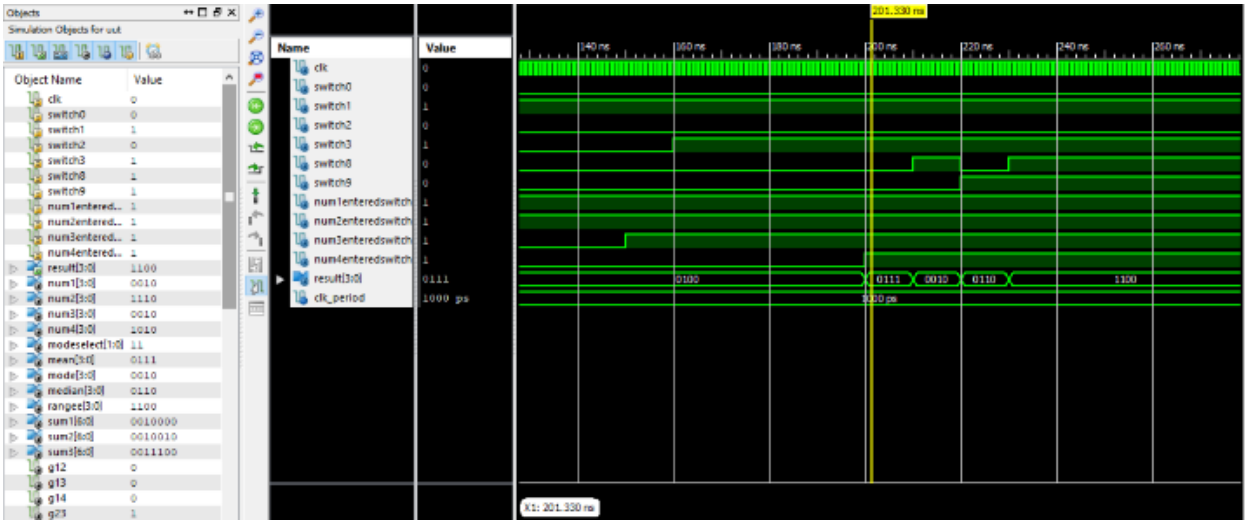
Since num2enteredswitch is turned on (became 1), the bits of num2 are assigned to the value switch3, switch2, switch1 and switch0, respectively. Therefore num2=1110 (14 in decimal):



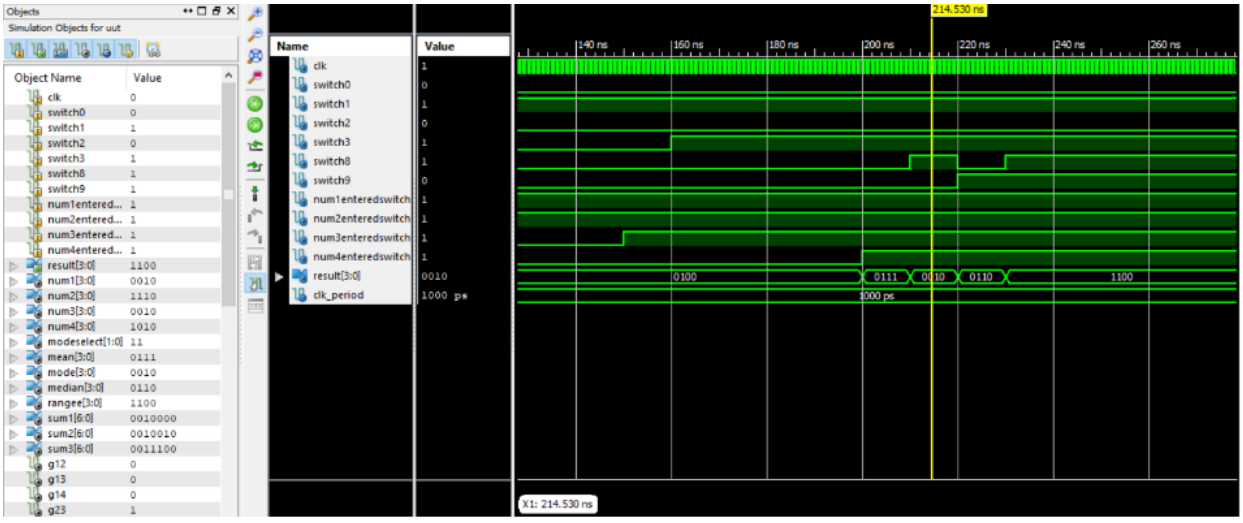
Since num3enteredswitch is turned on (became 1), the bits of num3 are assigned to the value switch3, switch2, switch1 and switch0, respectively. Therefore num3=0010 (2 in decimal):



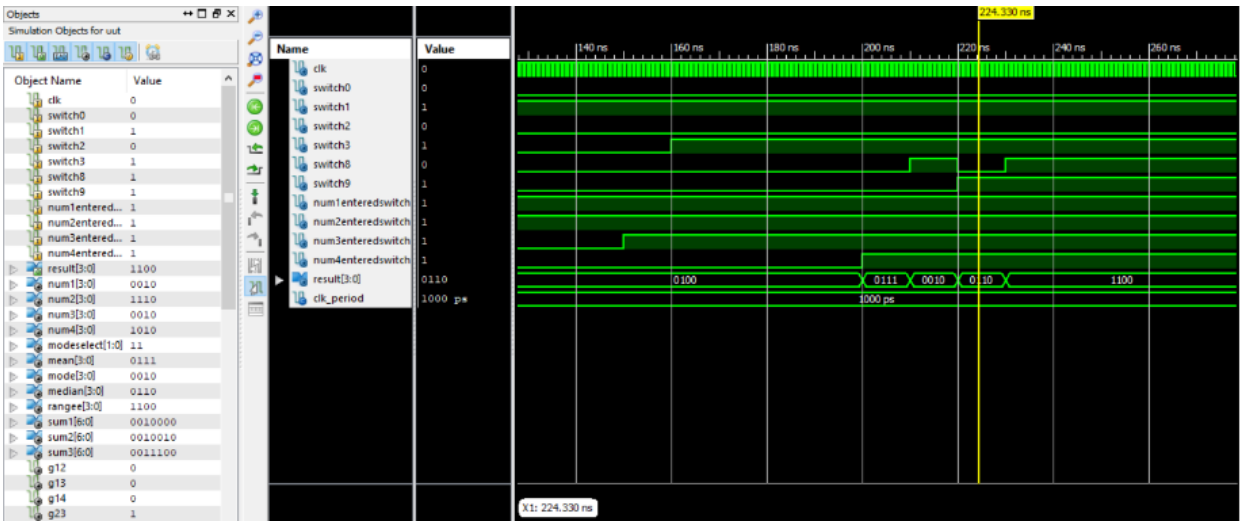
Since num4enteredswitch is turned on (became 1), the bits of num4 are assigned to the value switch3, switch2, switch1 and switch0, respectively. Therefore num4=1010 (10 in decimal). Then, since all the numbers are inputted and modeSelect=00 (switch9 = modeSelect(1) and switch8 = modeSelect(1)), mean (0111) is calculated and result is assigned to mean value.



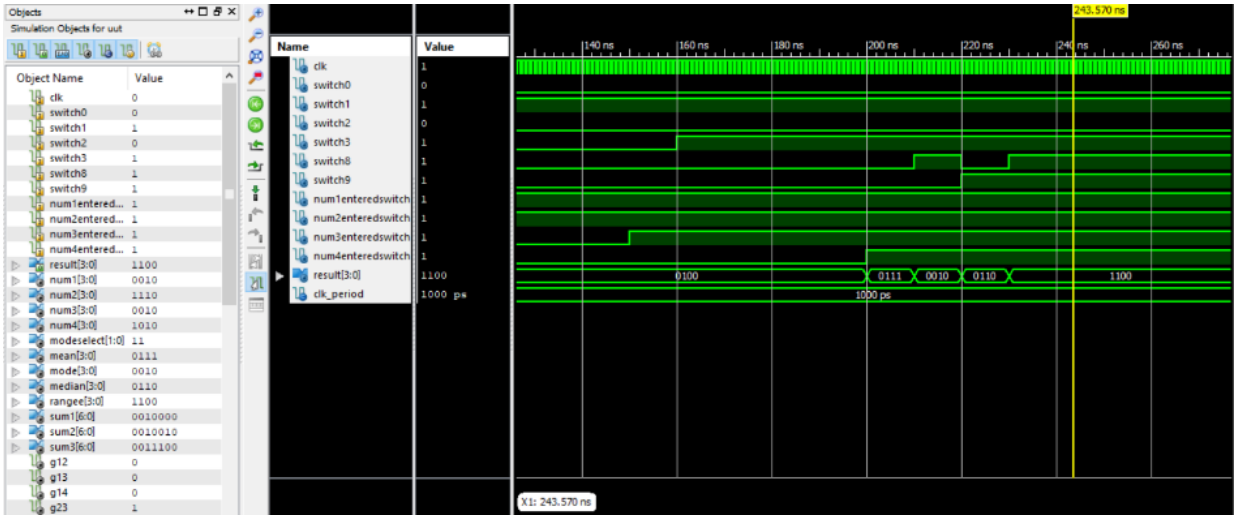
Since modeSelect=01, mode (0011) is calculated and result is assigned to mode value:



Since modeSelect=10, median (0110) is calculated and result is assigned to median value:



Since modeSelect=11, range (1100) is calculated and result is assigned to range value.



SUMMARY OF SIMULATION PHOTOS:

INPUTTED NUMBERS:

num1: 0010 (2 in decimal)
num2: 1110 (14 in decimal)
num3: 0010 (2 in decimal)
num4: 1010 (10 in decimal)

- ⇨ Mean = 28/4 = 7 (0111 in binary)
- ⇨ Mode = 2 (0010 in binary)
- ⇨ Median = (2+10) / 2 = 6 (0110 in binary)
- ⇨ Range = 14-2 = 12 (1100 in binary)